# Program Reasoning

7. Hoare Logic

Kihong Heo

Program Verification

- Specifying and proving properties of programs

- Specification: precise statement of properties that a program should exhibit in FOL

- Partial correctness properties: certain states cannot ever occur during the execution

  - "Bad things never happen" (e.g., integer overflow, buffer overflow, deadlock, etc)

  - Proof by inductive assertion method

- Total correctness properties: certain states are eventually reached during the execution

  - "Good things will eventually happen" (e.g., termination, fairness)

  - Proof by ranking function method

Overview
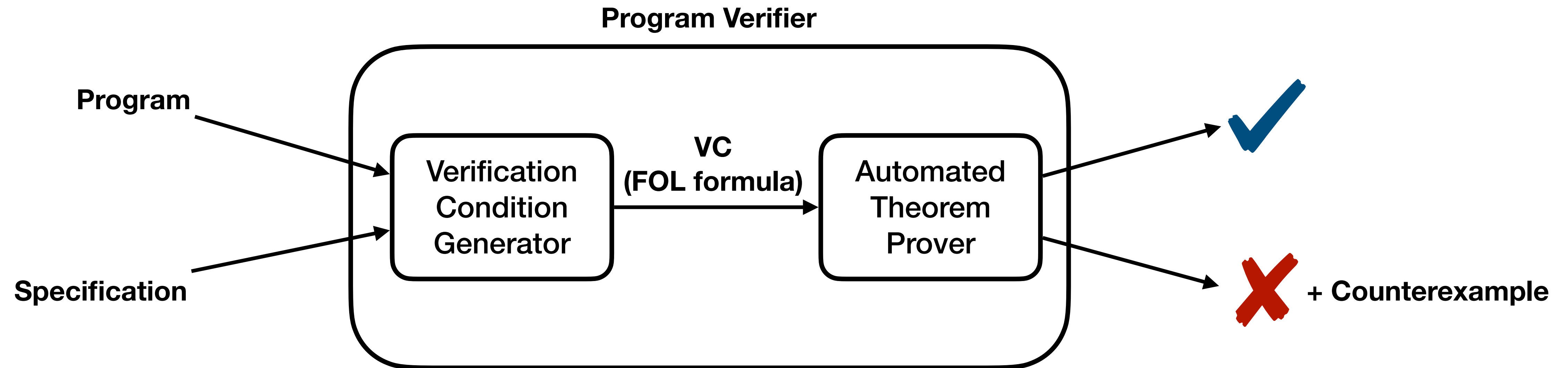
Specification

- Typically embedded into program text as program annotations

- An annotation is a FOL formula F

- An annotation F at location L asserts that F is true whenever program control reaches L

- Tree types of annotations:

  - Function specification

  - Loop invariant

  - Assertion

Function Specification

- A pair of annotations: precondition and postcondition

- Precondition: a formula whose free variables include only the formal parameters

  - "What should be true upon entering the function?"

- Postcondition: a formula whose free variables include only the formal parameters and the return value

  - "What is the relationship between the input and output?"

Example: Linear Search

- What would be the pre and post conditions?

  @pre: $0 \leq l \wedge u < |a|$

  @post: $rv \leftrightarrow \exists i . l \leq i \leq u \wedge a[i] = e$

  ```
  bool LinearSearch(int[] a, int l, int u, int e) {
    for (int i := l; i <= u; i := i + 1) {
      if (a[i] = e) return true;
    }
    return false;
  }
  ```

- BTW, is this nontrivial precondition (a formula other than $\top$) is always acceptable?

  - In terms of the software engineering practice (e.g., public API)

Example: More Robust Linear Search

- What would be the pre and post conditions?

@pre: $\top$

@post: $rv \leftrightarrow \exists i . l \le i \le u < |a| \wedge a[i] = e$

```
bool LinearSearch(int[] a, int l, int u, int e) {
  if (l < 0 \/ u >= |a|) return false;
  for (int i := l; i <= u; i := i + 1) {
    if (a[i] = e) return true;
  }
  return false;
}
```

Example: Binary Search

- What would be the pre and post conditions?

@pre: $0 \leq l \wedge u < |a| \wedge \text{sorted}(a, l, u)$

@post: $rv \leftrightarrow \exists i . l \leq i \leq u \wedge a[i] = e$

```
bool BinarySearch(int[] a, int l, int u, int e) {
  if (l > u) return false;
  int m := (l + u) / 2;
  if (a[m] = e) return true;
  else if (a[m] < e) return BinarySearch(a, m + 1, u, e)
  else return BinarySearch(a, l, m − 1, e)
}
```

- The sorted predicate is defined in the combined theory of integers and arrays:

$$\text{sorted}(a, l, u) \iff \forall i, j . l \leq i \leq j \leq u \rightarrow a[i] \leq a[j]$$

Example: Bubble Sort

- What would be the pre and post conditions?

@pre: $\top$

@post: $\text{sorted}(rv, 0, |rv| - 1)$

```
bool BubbleSort(int[] a0) {
  int a[] := a0;
  for (int i := |a| - 1; i > 0; i := i - 1) {
    for (int j := 0; j < i; j := j + 1) {
      if (a[j] > a[j + 1]) {
        int t := a[j];
        a[j] := a[j + 1];
        a[j + 1] := t;
      }
    }
  }
  return a;
}
```

Enough?

Loop Invariants

- Each loop has an annotation called loop invariant

$$\texttt{while}$$
$$@F$$
$$(\langle condition \rangle) \; \{$$
$$\langle body \rangle$$
$$\}$$

- The assertion F must hold at the beginning of every iteration

  - $F \wedge \langle condition \rangle$ holds on entering the body

  - $F \wedge \neg \langle condition \rangle$ holds when exiting the loop

- Why are loop invariants needed?

Example: Linear Search

- What would be the loop invariant?

@pre: $0 \leq l \wedge u < |a|$

@post: $rv \leftrightarrow \exists i . l \leq i \leq u \wedge a[i] = e$

```
bool LinearSearch(int[] a, int l, int u, int e) {
  int i := l;
  while
```
$$@L : l \leq i \wedge (\forall j . l \leq j < i \rightarrow a[j] \neq e)$$
```
    (i <= u) {
    if (a[i] = e) return true;
    i := i + 1;
  }
  return false;
}
```

Assertions

- Allows programmers to provide a formal comment

- Runtime assertions: a special class of assertions

  - E.g., division by 0, null dereference, etc

- Example: linear search with runtime assertions

```
bool LinearSearch(int[] a, int l, int u, int e) {
  int i := l;
  while (i <= u) {
    @ 0 ≤ i < |a|
    if (a[i] = e) return true;
    i := i + 1;
  }
  return false;
}
```

Inductive Assertion Method

- Proof technique for partial correctness of programs

- Idea: derive verification conditions from a function given annotations

- Example:

@precond S1 → S2 → S3 @postcond

@loopinv
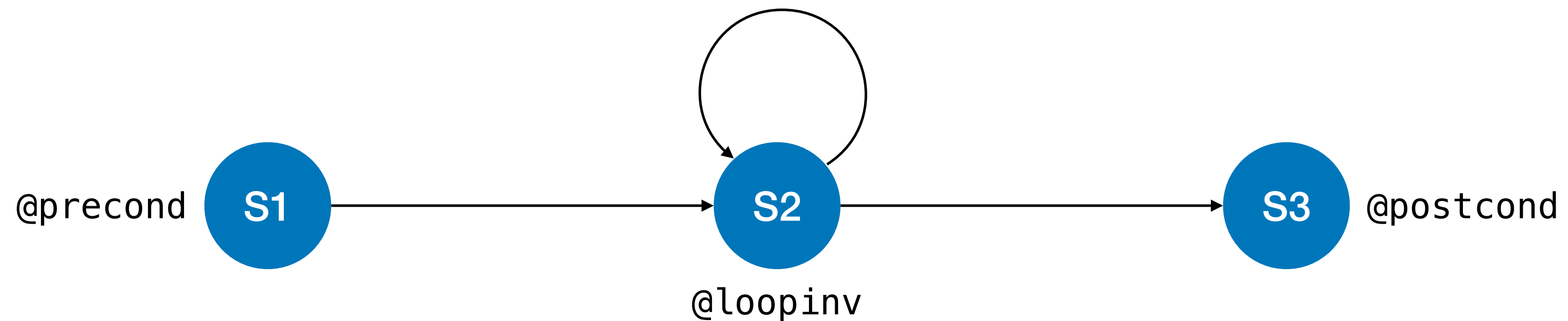
Hoare Triple

- Partial correctness specified using Hoare triple: $\{P\}\ S\ \{Q\}$

  - S : program fragment

  - P : precondition

  - Q : postcondition

- Meaning of Hoare triple:

  - If S is executed in a state satisfying P and if the execution of S terminates

  - Then, the program state after S terminates satisfies Q

Example

@pre: $0 \leq l \wedge u < |a|$

@post: $rv \leftrightarrow \exists i . l \leq i \leq u \wedge a[i] = e$
```
      bool LinearSearch(int[] a, int l, int u, int e) {
1:      int i := l;
2:      while
3:        @L : l ≤ i ∧ (∀j. l ≤ j < i → a[j] ≠ e)
4:        (i <= u) {
5:        if (a[i] = e)
6:          return true;
7:        i := i + 1
8:      }
9:      return false;
      }
```

@pre

①

@L

④

⑤   ⑨

@post

⑥   ⑦
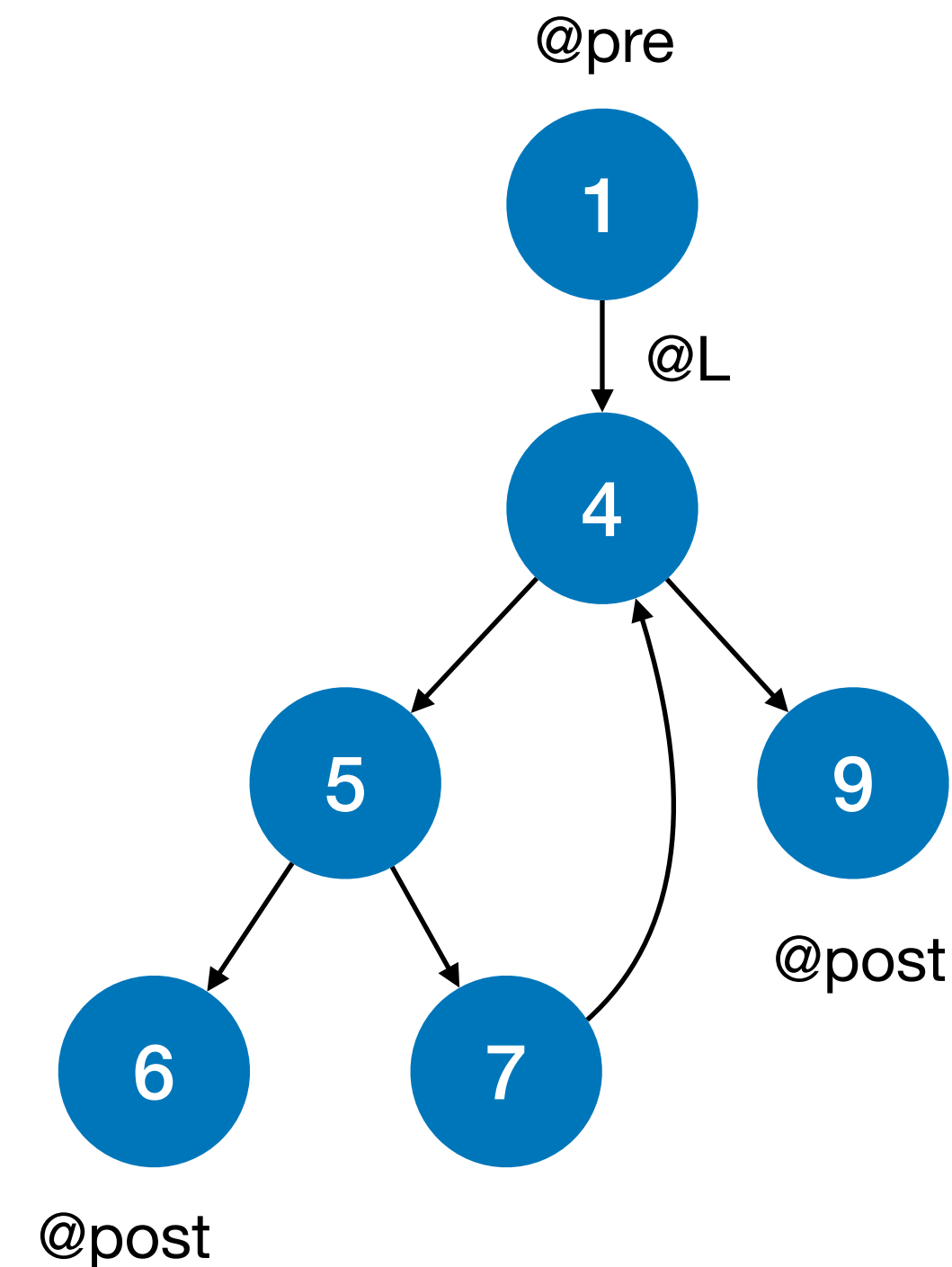
@post

- What conditions should be proven?

  {@pre} S1 {@L}   {@L} S4; S5; S6 {@post}   {@L} S4; S5; S7 {@L}   {@L} S4; S9 {@post}

- What about {@L} S4; S5; S7; S4; S5; S6 {@post} ? Why?

Hoare Logic

- A logic to prove the validity of Hoare triple

- A set of logical rules for reasoning about the partial correctness of programs

- In this lecture, we assume the following simple imperative language

$$S \rightarrow \mathtt{skip}$$
$$| \ x := E$$
$$| \ S; S$$
$$| \ \mathtt{if} \ E \ \mathtt{then} \ S \ \mathtt{else} \ S$$
$$| \ \mathtt{while} \ E \ \mathtt{do} \ S$$

Example

- Which one is valid?

  - $\{x = 0\}$ `x := x + 1` $\{x = 1\}$

  - $\{x = 0 \wedge y = 1\}$ `x := x + 1` $\{x = 1 \wedge y = 2\}$

  - $\{x = 0\}$ `x := x + 1` $\{x = 1 \vee y = 2\}$

  - $\{x = 0\}$ `while true do x := 0` $\{x = 1\}$

Hoare Rules (1)

- Rule for skip

$$\frac{}{\{P\} \ \texttt{skip} \ \{P\}}$$

- Rule for assignment

$$\frac{}{\{Q[E/x]\} \ x := E \ \{Q\}}$$

  - Intuition: revert to the state before the assignment

  - Example:

$$\{\texttt{true}\} \ x := 1 \ \{x = 1\} \qquad\qquad \{x + 1 > 0\} \ x := x + 1 \ \{x > 0\}$$

$$\{y = 1\} \ x := y \ \{x = 1\} \qquad\qquad \{\texttt{false}\} \ x := y + 3 \ \{y = 0 \wedge x = 12\}$$

Hoare Rules (2)

- Rule for precondition strengthening
$$\frac{\{P'\} \; S \; \{Q\} \quad P \implies P'}{\{P\} \; S \; \{Q\}}$$

  - Example:

$$\frac{\dfrac{\{y = x[x/y]\} \; y := x \; \{y = x\}}{\{\mathsf{true}\} \; y := x \; \{y = x\}} \quad z = 2 \implies \mathsf{true}}{\{z = 2\} \; y := x \; \{y = x\}}$$

- Rule for postcondition weakening
$$\frac{\{P\} \; S \; \{Q'\} \quad Q' \implies Q}{\{P\} \; S \; \{Q\}}$$

  - Example:

$$\frac{\dfrac{\dots}{\{\mathsf{true}\} \; S \; \{x = y \wedge z = 2\}} \quad x = y \wedge z = 2 \implies x = y}{\{\mathsf{true}\} \; S \; \{x = y\}}$$

Hoare Rules (3)

- Rule for composition

$$\frac{\{P\}\ S_1\ \{Q\} \quad \{Q\}\ S_2\ \{R\}}{\{P\}\ S_1; S_2\ \{R\}}$$

$$\frac{\dfrac{\{x = 2[2/x]\}\ x := 2\ \{x = 2\}}{\{\mathtt{true}\}\ x := 2\ \{x = 2\}} \quad \dfrac{\{x = 2 \land y = 2[x/y]\}\ y := x\ \{x = 2 \land y = 2\}}{\{x = 2\}\ y := x\ \{x = 2 \land y = 2\}}}{\{\mathtt{true}\}\ x := 2;\ y := x\ \{x = 2 \land y = 2\}}$$

- Rule for if statement

$$\frac{\{P \land E\}\ S_1\ \{Q\} \quad \{P \land \neg E\}\ S_2\ \{Q\}}{\{P\}\ \mathtt{if}\ E\ \mathtt{then}\ S_1\ \mathtt{else}\ S_2\ \{Q\}}$$

$$\frac{\dfrac{\{y \geq 0[x/y]\}\ y := x\ \{y \geq 0\} \quad x \geq 0 \implies x > 0}{\{x > 0\}\ y := x\ \{y \geq 0\}} \quad \dfrac{\{y \geq 0[-x/y]\}\ y := -x\ \{y \geq 0\}}{\{x \leq 0\}\ y := -x\ \{y \geq 0\}}}{\{\mathtt{true}\}\ \mathtt{if}\ x > 0\ \mathtt{then}\ y := x\ \mathtt{else}\ y := -x\ \{y \geq 0\}}$$

Hoare Rules (4)

- Rule for loop
$$\frac{\{P \wedge E\}\ S\ \{P\}}{\{P\}\ \texttt{while}\ E\ \texttt{do}\ S\ \{P \wedge \neg E\}}$$

$$\frac{\dfrac{\dots}{\{x \leq n \wedge x < n\}\ x := x + 1\ \{x \leq n\}}}{\{x \leq n\}\ \texttt{while}\ x < n\ \texttt{do}\ x := x + 1\ \{x \geq n\}}$$

Loop Invariant

- Challenge: impossible to know how many times a given loop iterates

- How to prove the partial correctness of a loop within finite time?

- Analogy: mathematical induction

- Loop invariant I satisfies the following properties:

  - I holds initially before the loop

  - I holds after each iteration of the loop

- Example
  ```
  i := 0; j := 0; n := 10;
  while (i < n) { // loop invariants?
    i := i + 1;
    j := i + j;
  }
  ```

Inductive Invariant

- Not all invariants are provable

- Example:

```
i := 5;
while (i > 1) { // invariant: i > 0
  i := i - 2;
}
assert(i = 1);
```

$$\frac{\{P \wedge E\}\ S\ \{P\}}{\{P\}\ \texttt{while}\ E\ \texttt{do}\ S\ \{P \wedge \neg E\}}$$

- Inductive invariant: invariant we can prove using induction

- Challenge: finding inductive loop invariants

  - Practice: human, static analysis, machine learning, etc

Automatically Proving Partial Correctness

- $\{P\}\ S\ \{Q\}$: Given the precondition satisfied, the postcondition is satisfied after the execution (if it terminates)

- Assumption: loop invariants are given by an oracle

  - Oracle: human, static analysis, machine learning, etc

- How to automatically prove correctness?

- Idea: deriving verification conditions (VCs) and check the validity

Verification Condition

- A FOL formula F such that the program is correct iff F is valid

- Automatically proving partial correctness

  - Generating VCs from a program + checking the validity of VCs by a theorem prover

- Two ways to generate verification conditions

  - Forward: starting from prediction, generate formulas to prove postcondition (strongest postconditions)

  - Backward: starting from postcondition, generate formulas to prove precondition (weakest preconditions)

Weakest Liberal Preconditions

- Goal: verify Hoar triple $\{P\}\ S\ \{Q\}$

- Weakest liberal precondition $wlp(S, Q)$ [Dijkstra75]

  - Weakest: most general condition that guarantees Q will hold after S in any execution

  - Liberal: we do not care about termination

- Proof of the Hoar triple $\{P\}\ S\ \{Q\}$: $P \to wlp(S, Q)$

- Example: $\{y \geq 10\}\ x := y + 1\ \{x \geq 0\}$
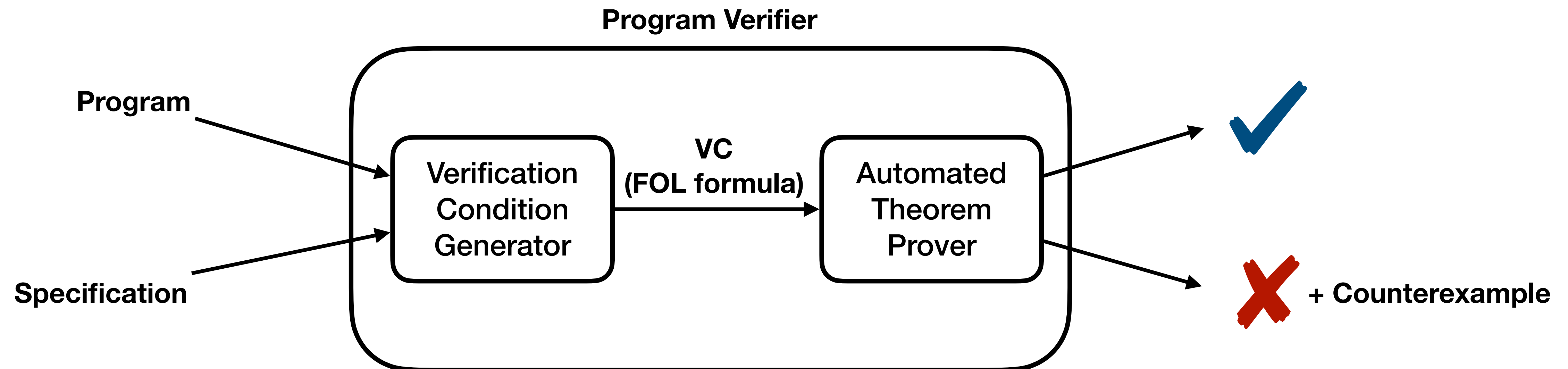
Weakest Precondition Calculus

- Inductively define wlp following Hoare rules

- $wlp(x := E, Q) = Q[E/x]$

- $wlp(s_1; s_2, Q) = wlp(s_1, wlp(s_2, Q))$

- $wlp(\text{if } E \text{ then } s_1 \text{ else } s_2, Q) = E \rightarrow wlp(s_1, Q) \wedge \neg E \rightarrow wlp(s_2, Q)$

- $wlp(\text{while } E \text{ do } S, Q) = I \wedge (E \wedge I \rightarrow wlp(S, I)) \wedge (\neg E \wedge I \rightarrow Q)$

  - Assumption: an inductive invariant I is provided

Example

- $S : x := y + 1; \text{if } x > 0 \text{ then } z := 1 \text{ else } z := -1$

- $wlp(S, z > 0)$?

- $wlp(S, z \leq 0)$?

- $\{y \geq -1\} \ S \ \{z = 1\}$?

- $\{y > -1\} \ S \ \{z = 1\}$?

Verification of Hoare Triple

- Validity of $\{P\}\ S\ \{Q\}$

- Verification condition: $P \rightarrow wlp(S, Q)$

Program Verifier

Program

Specification

Verification Condition Generator

VC (FOL formula)

Automated Theorem Prover

+ **Counterexample**

Summary

- Hoare triple: specifications for partial correctness $\{P\}\ S\ \{Q\}$

- Hoare logic: a logic to prove the validity of Hoare triple

    - Proof rules for each program command

- Verification condition is valid iff the Hoare triple is valid

- Automated program verification: check whether the VC is valid using theorem provers